



Java

В предыдущих сериях

- Java – язык с Си-подобным синтаксисом, автоматической сборкой мусора, часто используется для создания бизнес приложений
- JVM – виртуальная машина Java, на которой исполняется байт-код
- IDEA – среда разработки на Java
- Control flow на Java очень похоже на Си

Глава 3.1

Объектно- ориентированное программирование

ООП

- Человеку сложно воспринимать большие объемы процедурного кода
- ООП призвано упростить написание кода
- Идея ООП – структурирование программы в «модель реального мира»

ООП: идея

- Практически что угодно в нашем мире можно назвать объектом
- Объект можно охарактеризовать набором параметров
- Значения этих параметров можно считать «состоянием» объекта
- Кроме того, некоторые объекты умеют совершать действия

ООП: состояние и поведение

- Объекты умеют совершать какие - то действия
- Действия зависят от их состояния (не всегда, но часто)
- Эти действия называются методы
- Все, что умеет делать объект – его поведение.

ООП: пример

- Параметры объекта будем звать свойствами
- То, что он умеет делать – методами
- Какие свойства и методы есть у автомобиля?



ООП: класс

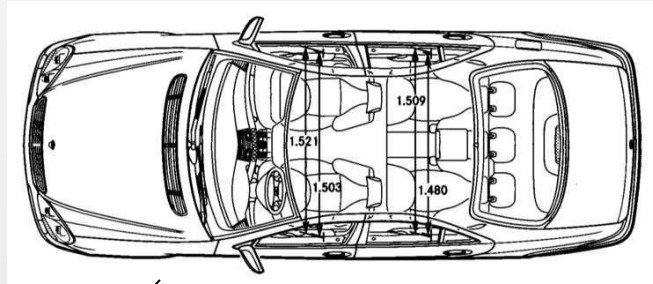
- Некий “чертеж” или “инструкция”, описывающая объект
- Описывает какие свойства есть у объекта (но не их конкретные значения)
- Описывает, что умеет делать объект, но поведение в «рантайме» часто зависит от свойств

ООП: основные понятия

- Объект – сущность, имеет состояние и поведение
- Класс – шаблон, по которому создается объект
- Свойство – параметр класса, у объекта имеет конкретное значение
- Метод – функция, описывается в классе, принадлежит объекту, может использовать его состояние
- Конструктор класса – специальный метод,

ООП: пример класса

Class Car



Object Volkswagen polo



Object Kia Shortage



Object BWM X6

ООП: пример класса

```
public class Car {  
  
    private String brand;  
    private Integer engineCapacity;  
  
    public Car(String brand, Integer engineCapacity) {  
        this.brand = brand;  
        this.engineCapacity = engineCapacity;  
    }  
  
    public void drive() {  
        System.out.println("я " + brand);  
        if ("Lada".equals(brand)) {  
            System.out.println("дж дж бр дзунь.. что то сломалось");  
            return;  
        }  
  
        if (engineCapacity > 150) {  
            System.out.println("вжууууух... я еду быстро");  
            return;  
        }  
        System.out.println("ж ж ж, я просто еду");  
    }  
}
```

Класс

Свойства

Конструктор
класса

Метод

```
public static void main(String[] args) {  
  
    Car lada = new Car("Lada", 90);  
    Car polo = new Car("Volkswagen", 110);  
    System.out.println("тестдрайв");  
    lada.drive();  
    polo.drive();  
}
```

```
"C:\Program Files\Java\jdk1.8.0_25\bin\java" ...
```

```
тестдрайв  
я Lada  
дж дж бр дзунь.. что то сломалось  
я Volkswagen  
ж ж ж, я просто еду
```

Добавим новый метод

```
public class Car {  
  
    private String brand;  
    private Integer engineCapacity;  
  
    public Car(String brand, Integer engineCapacity) {  
        this.brand = brand;  
        this.engineCapacity = engineCapacity;  
    }  
  
    public void drive() {  
        System.out.println("я " + brand);  
        if ("Lada".equals(brand)) {  
            System.out.println("дж дж бр дзунь.. что то сломалось");  
            return;  
        }  
  
        if (engineCapacity > 150) {  
            System.out.println("вжуууух... я еду быстро");  
            return;  
        }  
        System.out.println("ж ж ж, я просто еду");  
    }  
  
    public boolean upgradeEngine(int upgradeCapacity) {  
        if (engineCapacity + upgradeCapacity > 250) {  
            System.out.println("Неее, машина не выдержит");  
            return false;  
        }  
        this.engineCapacity += upgradeCapacity;  
        System.out.println("Тачка прокачалась, теперь у вас " + engineCapacity + " л.с.");  
        return true;  
    }  
}
```

```
package com.company;  
public class Main {  
  
    public static void main(String[] args) {  
  
        Car lada = new Car("Lada", 90);  
        Car polo = new Car("Volkswagen", 110);  
        System.out.println("тестдрайв");  
        lada.drive();  
        polo.drive();  
  
        System.out.println("прокачаем фолькс");  
        while(polo.upgradeEngine(60)) {  
            System.out.println("прокачаем ка еще разок");  
        }  
        polo.drive();  
    }  
}
```

```
"C:\Program Files\Java\jdk1.8.0_25\bin\java" ...
```

```
тестдрайв  
я Lada  
дж дж бр дзунь.. что то сломалось  
я Volkswagen  
ж ж ж, я просто еду
```

```
прокачаем фолькс  
Тачка прокачалась, теперь у вас 170 л.с.  
прокачаем ка еще разок  
Тачка прокачалась, теперь у вас 230 л.с.  
прокачаем ка еще разок  
Неее, машина не выдержит  
я Volkswagen  
вжуууух... я еду быстро
```

Вопросы и ответы



Глава 3.2

Принципы ООП

ДИСКЛЕЙМЕР

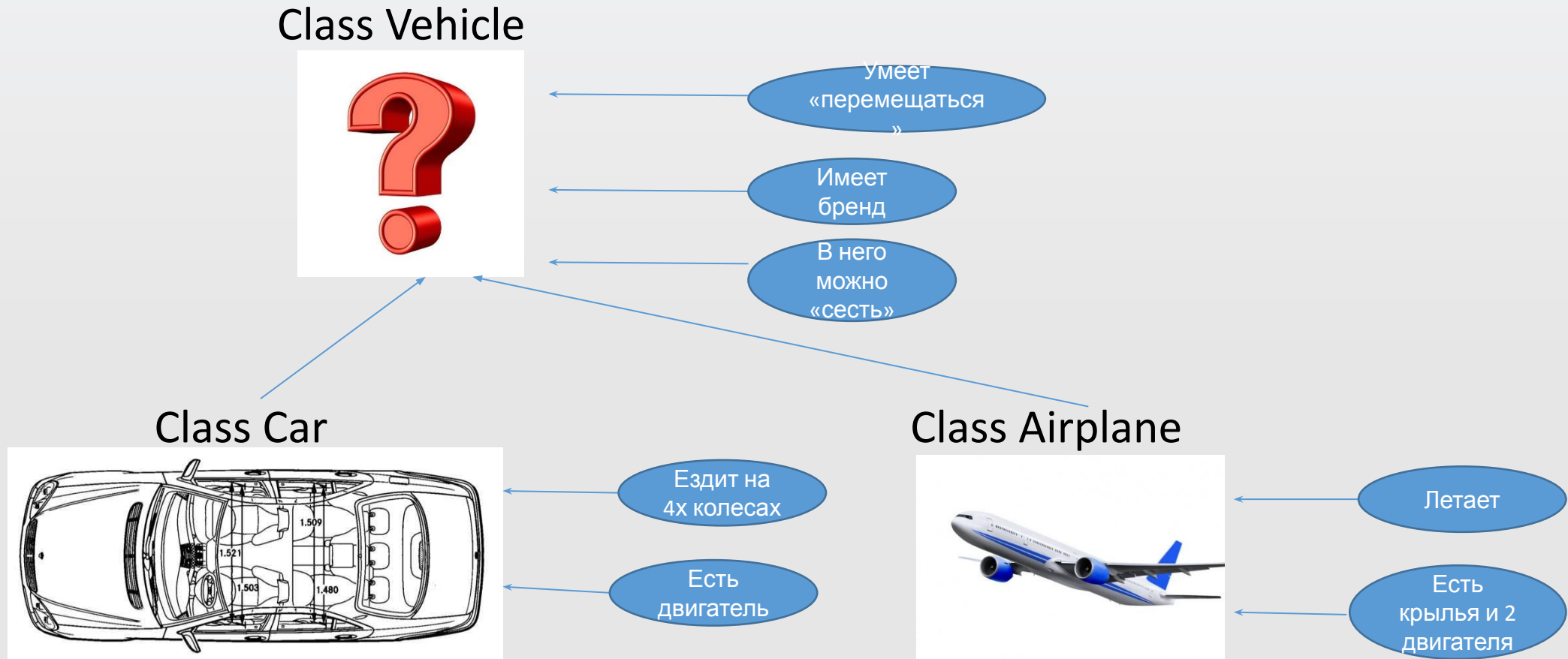
Есть общепризнанные концепции ООП, а есть моя интерпретация тех или иных свойств, и их применимость на практике. Часть из того, что я буду рассказывать, будет касаться личного опыта.

ООП: Принципы

- Полиморфизм
- Инкапсуляция
- Наследование

ООП: Наследование

- Вспомним класс «Машина»
- Подумаем о взаимоотношении класса «Машина» и «ТранспортноеСредство»



Наследование, пример

```
public class Vehicle {  
  
    protected String brand;  
    protected Integer passengersCount;  
  
    public Vehicle(String brand) {  
        this.brand = brand;  
        this.passengersCount = 0;  
    }  
  
    public void move(String destination) {  
        System.out.println("Я перемещаюсь в " + destination);  
    }  
  
    public void sitIn(Integer passengersCount) {  
        this.passengersCount += passengersCount;  
        System.out.println("Теперь у нас " + passengersCount + " пассажиров");  
    }  
  
}
```

ООП: Наследование, пример

```
public class Plane extends Vehicle {  
  
    private Integer engineCount = 2;  
    private Integer wingsCount = 2;  
  
    public Plane(String brand) {  
        super(brand);  
    }  
  
    @Override  
    public void move(String destination) {  
        if (wingsCount == 2 && engineCount == 2) {  
            System.out.println("Я лечу в " + destination);  
        } else {  
            System.out.println("Что то не так с самолетом");  
        }  
    }  
}
```

Наследование

Значение по умолчанию

Обращение к конструктору предка

Переопределил и метод

```
public class Car extends Vehicle {  
  
    private Integer engineCapacity;  
  
    public Car(String brand, Integer engineCapacity) {  
        super(brand);  
        this.engineCapacity = engineCapacity;  
    }  
  
    @Override  
    public void move(String destination) {  
        String speedDescription = engineCapacity > 100 ? "быстро" : "медленно";  
        System.out.println("Я перемещаюсь в " + destination + " очень " + speedDescription);  
    }  
}
```

Унарный условный оператор

ООП: Наследование, пример

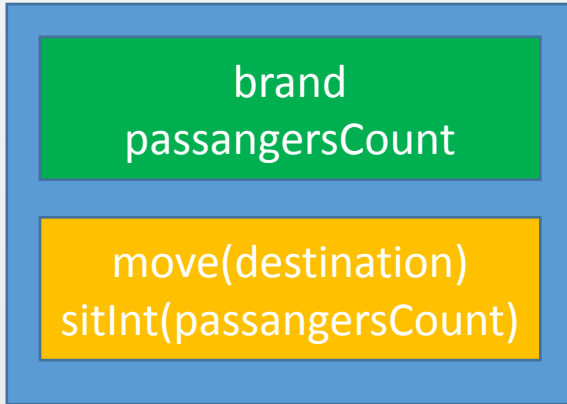
```
public class Main {  
  
    public static void main(String[] args) {  
        Car bmwCar = new Car( brand: "BMW", engineCapacity: 200);  
        Car toyotaCar = new Car( brand: "Toyota", engineCapacity: 80);  
        Plane plane = new Plane( brand: "Toyota");  
        bmwCar.sitIn( passengersCount: 2);  
        bmwCar.move( destination: "Светлое будущее");  
        toyotaCar.move( destination: "Темное прошлое");  
        plane.move( destination: "Воронеж");  
    }  
}
```

```
C:\Users\kondo\.jdk\corretto-15.0.2\bin\java.exe "-javaagent:C:\Pr  
Теперь у нас 2 пассажиров  
Я перемещаюсь в Светлое будущее очень быстро  
Я перемещаюсь в Темное прошлое очень медленно  
Я лечу в Воронеж  
Process finished with exit code 0
```

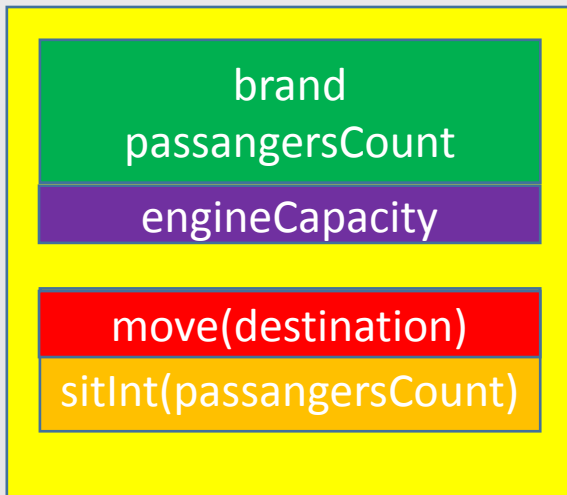
Совсем чуть-чуть про память в

java

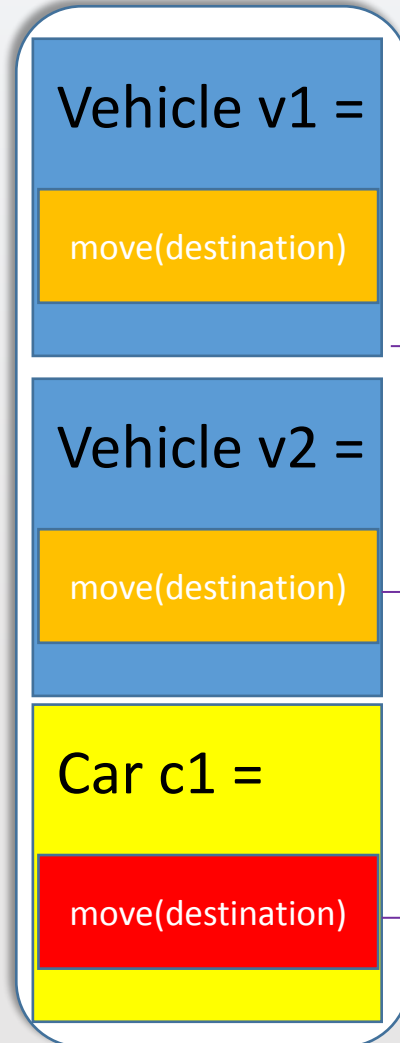
Class Vehicle



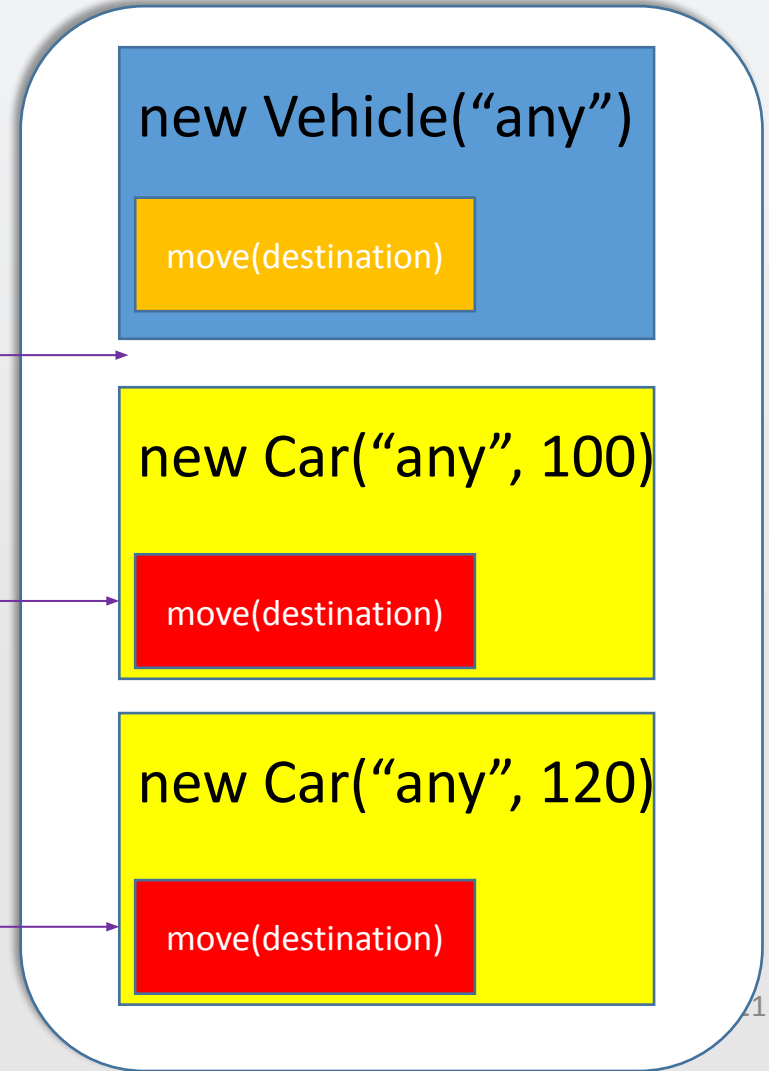
Class Car



Stack



Heap



ООП: Инкапсуляция

- Упаковка данных и функций в единый компонент
- Скрывает детали реализации от пользователя класса
- Делает систему более гибкой

ООП: Инкапсуляция, модификаторы доступа

- Public – виден всем
- Protected – виден в потомках
- Private – виден только самому классу
- Package Private – виден в рамках пакета

Инкапсуляция, пример

```
public class Plane extends Vehicle {
```

```
    private Integer engineCount = 2;  
    private Integer wingsCount = 2;
```

Поле
недоступно из
вне

```
    public Plane(String brand) {  
        super(brand);  
    }
```

```
@Override
```

```
public void move(String destination) {  
    if (wingsCount == 2 && engineCount == 2) {  
        System.out.println("Я лечу в " + destination);  
    } else {  
        System.out.println("Что то не так с самолетом");  
    }  
}
```

Метод доступен
всем

Поле доступно
внутри класса

```
public class Main {
```

```
    public static void main(String[] args) {  
        Plane plane = new Plane( brand: "Toyota");  
        plane.wingsCount = -3;  
        plane.move( destination: "Воронеж");  
    }  
}
```

```
C:\Users\kondo\IdeaProjects\test\src\main\java\Main.java:7:14  
java: wingsCount has private access in vehicle.Plane
```


Инкапсуляция, пример

```
public class Plane extends Vehicle {  
  
    private Integer engineCount = 2;  
    public Integer wingsCount = 2;  
  
    public Plane(String brand) {  
        super(brand);  
    }  
  
    @Override  
    public void move(String destination) {  
        if (wingsCount == 2 && engineCount == 2) {  
            System.out.println("Я лечу в " + destination);  
        } else {  
            System.out.println("Что то не так с самолетом");  
        }  
    }  
}
```

Нарушаем
инкапсуляцию

```
public class Main {  
  
    public static void main(String[] args) {  
        Plane plane = new Plane( brand: "Toyota");  
        plane.wingsCount = -3;  
        plane.move( destination: "Воронеж");  
    }  
}
```

```
Main x  
C:\Users\kondo\.jdk\corretto-15.0.2\bin\jav  
Что то не так с самолетом  
  
Process finished with exit code 0
```

Инкапсуляция, пример

```
public class Vehicle {  
  
    protected String brand;  
    protected Integer passengersCount;  
  
    public Vehicle(String brand) {  
        this.brand = brand;  
        this.passengersCount = 0;  
    }  
  
    public void move(String destination) {  
        System.out.println("Я перемещаюсь в " + destination);  
    }  
  
    public void sitIn(Integer passengersCount) {  
        this.passengersCount += passengersCount;  
        System.out.println("Теперь у нас " + passengersCount + " пассажиров");  
    }  
}
```

Поле доступно
в потомках

```
public class Car extends Vehicle {  
  
    private Integer engineCapacity;  
  
    public Car(String brand, Integer engineCapacity) {  
        super(brand);  
        this.engineCapacity = engineCapacity;  
    }  
  
    @Override  
    public void move(String destination) {  
        String speedDescription = engineCapacity > 100 ? "быстро" : "медленно";  
        System.out.println("Я " + brand + " перемещаюсь в " + destination + " очень " + speedDescription);  
    }  
}
```

ООП: полиморфизм

- Некоторые объекты могут совершать похожие действия
- Транспортные средства могут ездить, но все по-разному
- С точки зрения ООП – наличие у объектов-потомков методов, одинаковых по сигнатуре, но различных по реализации
- Позволяют использовать похожие алгоритмы с различной реализацией в одном контексте.

ООП: полиморфизм

```
public class Main {  
  
    public static void main(String[] args) {  
        Car bmwCar = new Car( brand: "BMW", engineCapacity: 200);  
        Car toyotaCar = new Car( brand: "Toyota", engineCapacity: 80);  
        Plane plane = new Plane( brand: "Tu134");  
  
        Vehicle[] vehicles = {bmwCar, toyotaCar, plane};  
        for (Vehicle vehicle : vehicles) {  
            vehicle.move( destination: "Воронеж");  
        }  
    }  
}
```

Полиморфизм

Код не зависит
от типа
транспорта

```
Main x  
C:\Users\kondo\.jdk\corretto-15.0.2\bin\java.exe "-ja  
Я BMW перемещаюсь в Воронеж очень быстро  
Я Toyota перемещаюсь в Воронеж очень медленно  
Я лечу в Воронеж  
finished with exit code 0
```

ООП: оверлоад

```
return 0;
}

int main(int argc, char* argv[]) {
    Math.|
    m random() double
    m atan2(double y, double x) double
    m max(int a, int b) int
    m max(long a, long b) long
    m max(float a, float b) float

```

Зачем вообще нужно ООП?



исмаил с ответил Полине

это обман чтобы набрать классы

👍 Класс! 1



Зачем вообще нужно ООП?

- Код разбивается на большее число модулей
- Модули менее зависимы
- Связи между модулями не зависят от реализации

Глава 3.3

Абстрактный класс и интерфейс

Абстрактный класс

- Метод `move` всегда переопределяется
- Мы никогда не используем `Vehicle`, только наследников

```
public class Vehicle {  
  
    protected String brand;  
    protected Integer passengersCount;  
  
    public Vehicle(String brand) {  
        this.brand = brand;  
        this.passengersCount = 0;  
    }  
  
    public void move(String destination) {  
        System.out.println("Я перемещаюсь в " + destination);  
    }  
  
    public void sitIn(Integer passengersCount) {  
        this.passengersCount += passengersCount;  
        System.out.println("Теперь у нас " + passengersCount + " пассажиров");  
    }  
  
}
```

Абстрактный класс

- Абстрактный класс – класс, у которого есть хотя бы 1 метод (вообще может и не быть, но смысла в этом мало), который непонятно как должен работать, и по-этому определяться он будет в наследниках
- Сущность абстрактного класса создать нельзя (но можно присвоить наследника переменной ТИПА абстрактного класса:
`Abstact a = new Naslednic()`)

Абстрактный класс

```
public abstract class Vehicle {  
  
    protected String brand;  
    protected Integer passengersCount;  
  
    public Vehicle(String brand) {  
        this.brand = brand;  
        this.passengersCount = 0;  
    }  
  
    public abstract void move(String destination);  
  
    public void sitIn(Integer passengersCount) {  
        this.passengersCount += passengersCount;  
        System.out.println("Теперь у нас " + passengersCount + " пассажиров");  
    }  
}
```

Абстрактный класс

Абстрактный метод

Интерфейс

- Интерфейс – так сказать “контракт” на то, что класс реализует набор методов.
- До 2014 года(до java 8) интерфейс не мог иметь реализации методов
- Класс может реализовывать несколько интерфейсов (а наследовать абстрактный класс только 1)

Интерфейс

```
public interface Movable {  
  
    void move(String destination);  
  
}
```

```
public interface PassengerTransport {  
  
    void sitIn(Integer passengersCount);  
  
}
```

```
public abstract class Vehicle implements Movable, PassengerTransport {  
  
    protected String brand;  
    protected Integer passengersCount;  
  
    public Vehicle(String brand) {  
        this.brand = brand;  
        this.passengersCount = 0;  
    }  
  
    public abstract void move(String destination);  
  
    public void sitIn(Integer passengersCount) {  
        this.passengersCount += passengersCount;  
        System.out.println("Теперь у нас " + passengersCount + " пассажиров");  
    }  
  
}
```

Множественное наследование интерфейсов

```
public class Main {  
  
    public static void main(String[] args) {  
        Plane plane = new Plane( brand: "Ty134");  
        Movable movable = plane;  
        PassengerTransport passengerTransport = plane;  
        passengerTransport.sitIn( passengersCount: 3);  
        movable.move( destination: "Воронеж");  
    }  
  
}
```

Приведение к типу интерфейса

Вопросы и ответы



Глава 3.4

Практика, Основы ООП

Практика

- Придумать иерархию классов
- Использовать все 3 принципа ООП
- Использовать абстрактный класс
- Продемонстрировать полиморфизм, используя вызов методов потомков через ссылки на предков